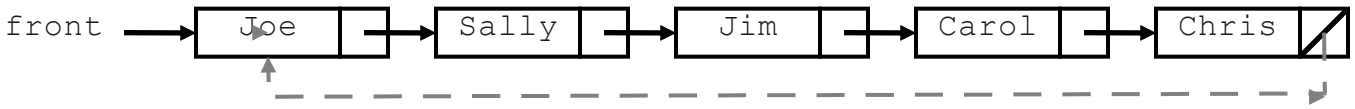
CSE 143, Winter 2011
Programming Assignment #4: Assassin (40 points)
Due Thursday, February 3, 2010, 11:30 PM

This program focuses on implementing a linked list. Turn in a file named `AssassinManager.java` from the Homework section of the course web site. You will need the support files `AssassinNode.java`, `AssassinMain.java`, and `names.txt` from the Homework section of the course web site; place them in the same folder as your class.

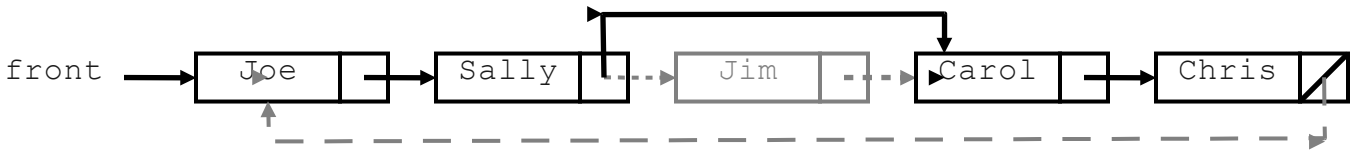
Program Description:

"Assassin" is a game often played on college campuses. Each person playing has a particular target that he/she is trying to "assassinate." Generally "assassinating" a person means finding them on campus in public and acting on them in some way, such as saying, "You're dead," or squirting them with a water gun, or touching them. One of the things that makes the game more interesting to play in real life is that initially each person knows only who they are assassinating; they don't know who is trying to assassinate them, nor do they know whom the other people are trying to assassinate.

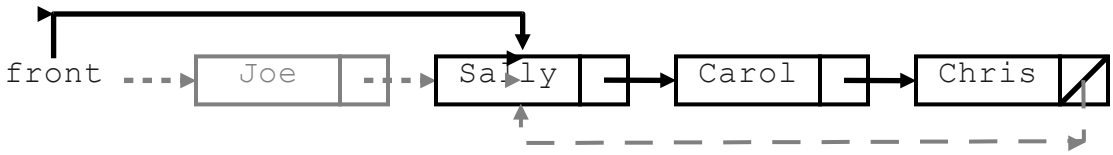
The game of assassin is played as follows: You start out with a group of people who want to play the game. For example, let's say that there are five people playing named Carol, Chris, Jim, Joe, Sally. A circular chain of assassination targets (called the "kill ring" in this program) is randomly established. For example, we might decide Joe should stalk Sally, Sally should stalk Jim, Jim should stalk Carol, Carol should stalk Chris, and Chris should stalk Joe. (In the actual linked list that implements this kill ring, Chris's `next` reference would be `null`. But conceptually we can think of it as though the next person after Chris is Joe, the front person in the list.) Here is a picture of this "kill ring":



When someone is assassinated, the links need to be changed to "skip" that person. For example, suppose that Jim is assassinated by Sally. Sally needs a new target, so we give her Jim's target: Carol. The kill ring becomes:



If the first person in the kill ring is assassinated, the front of the list must adjust. If Chris kills Joe, the list becomes:



You will write a class `AssassinManager` that keeps track of who is stalking whom and the history of who killed whom. You will maintain two linked lists: a list of people currently alive (the "kill ring") and a list of those who are dead (the "graveyard"). As people are killed, you will move them from the kill ring to the graveyard by rearranging links between nodes. The game ends when only one node remains in the kill ring, representing the winner.

A client program has been written for you called `AssassinMain`. It reads a file of names, shuffles the names, and constructs an object of your class `AssassinManager`. This main program then asks the user for the names of each victim to kill until there is just one player left alive (at which point the game is over and the last remaining player wins). `AssassinMain` calls methods of the `AssassinManager` class to carry out the tasks involved in administering the game.

Sample Log of Execution (user input underlined):

Your program should exactly reproduce the format and general behavior demonstrated in this log, although you may not exactly recreate this scenario because of the shuffling of the names that `AssassinMain` performs.

```
Current kill ring:
  Erica Kane is stalking Ruth Martin
  Ruth Martin is stalking Jackson Montgomery
  Jackson Montgomery is stalking Bobby Warner
  Bobby Warner is stalking Joe Martin
  Joe Martin is stalking Anita Santos
  Anita Santos is stalking Tad Martin
  Tad Martin is stalking Phoebe Wallingford
  Phoebe Wallingford is stalking Erica Kane
Current graveyard:

next victim? Ruth Martin

Current kill ring:
  Erica Kane is stalking Jackson Montgomery
  Jackson Montgomery is stalking Bobby Warner
  Bobby Warner is stalking Joe Martin
  Joe Martin is stalking Anita Santos
  Anita Santos is stalking Tad Martin
  Tad Martin is stalking Phoebe Wallingford
  Phoebe Wallingford is stalking Erica Kane
Current graveyard:
  Ruth Martin was killed by Erica Kane

next victim? Ruth Martin
Ruth Martin is already dead.

Current kill ring:
  Erica Kane is stalking Jackson Montgomery
  Jackson Montgomery is stalking Bobby Warner
  Bobby Warner is stalking Joe Martin
  Joe Martin is stalking Anita Santos
  Anita Santos is stalking Tad Martin
  Tad Martin is stalking Phoebe Wallingford
  Phoebe Wallingford is stalking Erica Kane
Current graveyard:
  Ruth Martin was killed by Erica Kane

next victim? bobby warner

Current kill ring:
  Erica Kane is stalking Jackson Montgomery
  Jackson Montgomery is stalking Joe Martin
  Joe Martin is stalking Anita Santos
  Anita Santos is stalking Tad Martin
  Tad Martin is stalking Phoebe Wallingford
  Phoebe Wallingford is stalking Erica Kane
Current graveyard:
  Bobby Warner was killed by Jackson Montgomery
  Ruth Martin was killed by Erica Kane

next victim? ERICa kaNE

Current kill ring:
  Jackson Montgomery is stalking Joe Martin
  Joe Martin is stalking Anita Santos
  Anita Santos is stalking Tad Martin
  Tad Martin is stalking Phoebe Wallingford
  Phoebe Wallingford is stalking Jackson Montgomery
Current graveyard:
  Erica Kane was killed by Phoebe Wallingford
  Bobby Warner was killed by Jackson Montgomery
  Ruth Martin was killed by Erica Kane

next victim? ANITA SANTOS
```

(continued)

```
Current kill ring:
  Jackson Montgomery is stalking Joe Martin
  Joe Martin is stalking Tad Martin
  Tad Martin is stalking Phoebe Wallingford
  Phoebe Wallingford is stalking Jackson Montgomery
Current graveyard:
  Anita Santos was killed by Joe Martin
  Erica Kane was killed by Phoebe Wallingford
  Bobby Warner was killed by Jackson Montgomery
  Ruth Martin was killed by Erica Kane

next victim? phoebe wallingford

Current kill ring:
  Jackson Montgomery is stalking Joe Martin
  Joe Martin is stalking Tad Martin
  Tad Martin is stalking Jackson Montgomery
Current graveyard:
  Phoebe Wallingford was killed by Tad Martin
  Anita Santos was killed by Joe Martin
  Erica Kane was killed by Phoebe Wallingford
  Bobby Warner was killed by Jackson Montgomery
  Ruth Martin was killed by Erica Kane

next victim? Barack Obama
Unknown person.

Current kill ring:
  Jackson Montgomery is stalking Joe Martin
  Joe Martin is stalking Tad Martin
  Tad Martin is stalking Jackson Montgomery
Current graveyard:
  Phoebe Wallingford was killed by Tad Martin
  Anita Santos was killed by Joe Martin
  Erica Kane was killed by Phoebe Wallingford
  Bobby Warner was killed by Jackson Montgomery
  Ruth Martin was killed by Erica Kane

next victim? Joe Martin

Current kill ring:
  Jackson Montgomery is stalking Tad Martin
  Tad Martin is stalking Jackson Montgomery
Current graveyard:
  Joe Martin was killed by Jackson Montgomery
  Phoebe Wallingford was killed by Tad Martin
  Anita Santos was killed by Joe Martin
  Erica Kane was killed by Phoebe Wallingford
  Bobby Warner was killed by Jackson Montgomery
  Ruth Martin was killed by Erica Kane

next victim? jackson montgomery

Game was won by Tad Martin
Final graveyard is as follows:
  Jackson Montgomery was killed by Tad Martin
  Joe Martin was killed by Jackson Montgomery
  Phoebe Wallingford was killed by Tad Martin
  Anita Santos was killed by Joe Martin
  Erica Kane was killed by Phoebe Wallingford
  Bobby Warner was killed by Jackson Montgomery
  Ruth Martin was killed by Erica Kane
```

Implementation Details:

You must use our node class `AssassinNode` (provided on the course website) which has the following implementation:

```
public class AssassinNode {
    public String name;          // this person's name
    public String killer;        // name of who killed this person (null if alive)
    public AssassinNode next;    // next node in the list

    public AssassinNode(String name) { ... }
    public AssassinNode(String name, AssassinNode next) { ... }
}
```

For this assignment we are going to specify exactly what fields you can have in your `AssassinManager` class. You must have exactly the following two fields; you are not allowed to have any others:

- a reference to the front node of the kill ring
- a reference to the front node of the graveyard (`null` if empty)

Implementation Details (continued):

Your class should have the following public methods.

`public AssassinManager(ArrayList<String> names)`

In this constructor you should initialize a new assassin manager over the given list of people. Your constructor should not save the list parameter itself as a field, nor modify the list; but instead it should build your own kill ring of linked nodes that contains these names in the same order. For example, if the list contains `["John", "Sally", "Fred"]`, the initial kill ring should represent that John is stalking Sally who is stalking Fred who is stalking John (in that order). You may assume that the names are non-empty, non-`null` strings and that there are no duplicates.

You should throw an `IllegalArgumentException` if the list is `null` or has a size of 0.

`public void printKillRing()`

In this method you should print the names of the people in the kill ring, one per line, indented by two spaces, as "**name** is stalking **name**". The behavior is unspecified if the game is over. For the names on the first page, the initial output is:

```
  Joe is stalking Sally
  Sally is stalking Jim
  Jim is stalking Carol
  Carol is stalking Chris
  Chris is stalking Joe
```

`public void printGraveyard()`

In this method you should print the names of the people in the graveyard, one per line, with each line indented by two spaces, with output of the form "**name** was killed by **name**". It should print the names in the opposite of the order in which they were killed (most recently killed first, then next more recently killed, and so on). It should produce no output if the graveyard is empty. For example, from the previous names, if Jim is killed, then Chris, then Carol, the output is:

```
  Carol was killed by Sally
  Chris was killed by Carol
  Jim was killed by Sally
```

`public boolean killRingContains(String name)`

In this method you should return `true` if the given name is in the current kill ring and `false` otherwise. It should ignore case in comparing names; for example, if passed `"salLY"`, it should match a node with a name of `"Sally"`.

`public boolean graveyardContains(String name)`

In this method you should return `true` if the given name is in the current graveyard and `false` otherwise. It should ignore case in comparing names; for example, if passed `"CaRoL"`, it should match a node with a name of `"Carol"`.

`public boolean isGameOver()`

In this method you should return `true` if the game is over (i.e., if the kill ring has just one person) and `false` otherwise.

`public String winner()`

In this method you should return the name of the winner of the game, or `null` if the game is not over.

`public void kill(String name)`

In this method you should record the killing of the person with the given name, transferring the person from the kill ring to the front of the graveyard. This operation should not change the relative order of the kill ring (i.e., the links of who is killing whom should stay the same other than the person who is being killed/removed). Ignore case in comparing names. A node remembers who killed the person in its `killer` field. It is your code's responsibility to set that field's value.

You should throw an `IllegalStateException` if the game is over, or an `IllegalArgumentException` if the given name is not part of the kill ring (if both of these conditions are true, the `IllegalStateException` takes precedence).

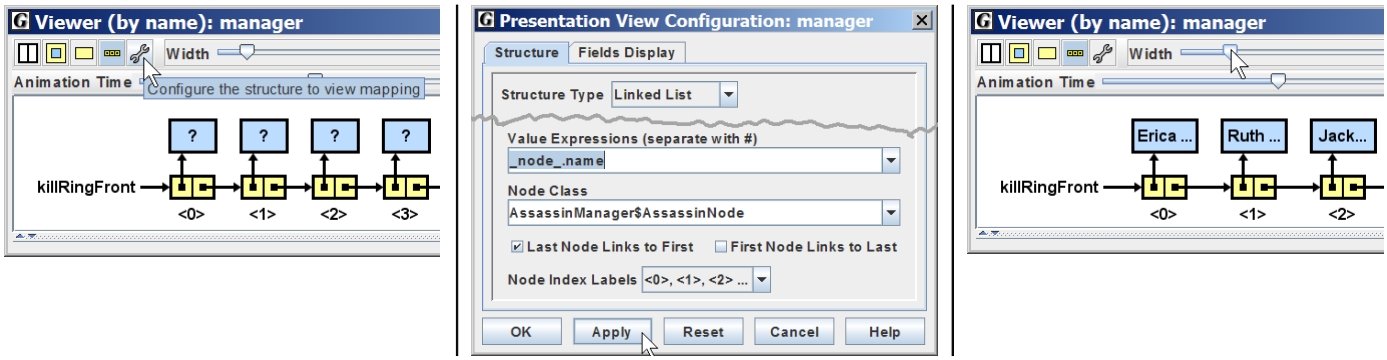
The `kill` method is the hardest one, so write it last. Use the jGRASP debugger and `println` statements liberally to debug problems in your code. You will likely have a lot of `NullPointerException` errors, infinite loops, etc. and will have a very hard time tracking them down unless you are comfortable with debugging techniques and jGRASP.

For full credit, every method's runtime should be at worst O(*N*), where *N* is the number of people in your linked lists.

jGRASP's Debugger:

In jGRASP's debugger you can use a structure viewer to see what your list looks like by dragging one of your fields from the debug window outside the window. By default the viewer won't show you the `name` in each node (it'll show a "?" mark instead). Fix this by clicking the wrench icon, then in the "Value Expressions" box, type: **`_node_.name`**

Click OK, and you should see the names in the nodes. You can also drag the Width scrollbar to see the names better.



Other Constraints and Tips:

This is meant to be an exercise in linked list manipulation. As a result, you must adhere to the following rules:

- You must use our `AssassinNode` class for your lists. You are not allowed to modify it.
- You may not construct any arrays, `ArrayList`s, `LinkedList`s, stacks, queues, sets, maps, or other data structures; you must use linked nodes. You may not modify the list of `String`s passed to your constructor.
- If there are *N* names in the list of `String`s passed to your constructor, you should create exactly *N* new `AssassinNode` objects in your constructor. As people are killed, you have to move their node from the kill ring to the graveyard by changing references, without creating any new node objects.

Your constructor will create the initial kill ring of nodes, and then your class may not create any more nodes for the rest of the program. You are allowed to declare as many local variables of type `AssassinNode` (like `current` from lecture) as you like. `AssassinNode` variables are not node objects and therefore don't count against the limit of *n* nodes.

You should write some of your own testing code. `AssassinMain` requires every method to be written in order to compile, and it never generates any of the exceptions you have to handle, so it is not exhaustive. You may share your testing code with others on the message board so long as the code does not give away the details of your program solution.

A word of caution: Some students try to store the kill ring in a "circular" linked list, with the list's final element storing a `next` reference back to the front element. But we discourage you from implementing the program in this way; we strongly suggest that you follow the normal convention of having `null` in the `next` field of the last node. Most novices find it difficult to work with a circular list; it is easy to end up with infinite loops or other bugs. There is no need to use a circular list, because you can always get back to the front via the fields of your `AssassinManager`. If you feel strongly that you want to use a circular list, you are allowed to do so, but it is likely to make the program harder to write.

Style Guidelines and Grading:

Part of your grade will come from appropriately utilizing linked lists and nodes as described previously. Redundancy is another major grading focus; some methods are very similar, and you should avoid repeated logic as much as possible.

You should follow good general style guidelines such as: making fields `private` and avoiding unnecessary fields; appropriately using control structures like loops and `if`/`else` statements; properly using indentation, good variable names and proper types; and not having any lines of code longer than 100 characters. Do not use recursion on this assignment.

Comment your code descriptively in your own words at the top of your class, each method, and on complex sections of your code. Comments should explain each method's behavior, parameters, return, pre/post-conditions, and exceptions. For reference, our solution is around 130 lines long (60 "substantive" lines).